Отчёт по лабораторной работе №2

Операционные системы

Екатерина Павловна Канева

Содержание

# 1 Цель работы

* Изучить идеологию и применение средств контроля версий.
* Освоить умения по работе с git.

# 2 Выполнение лабораторной работы

## 2.1 Настройка GitHub

Предварительно мною уже был создан аккаунт на сайте https://github.com, а также была заполнена основная информация (рис. 1):

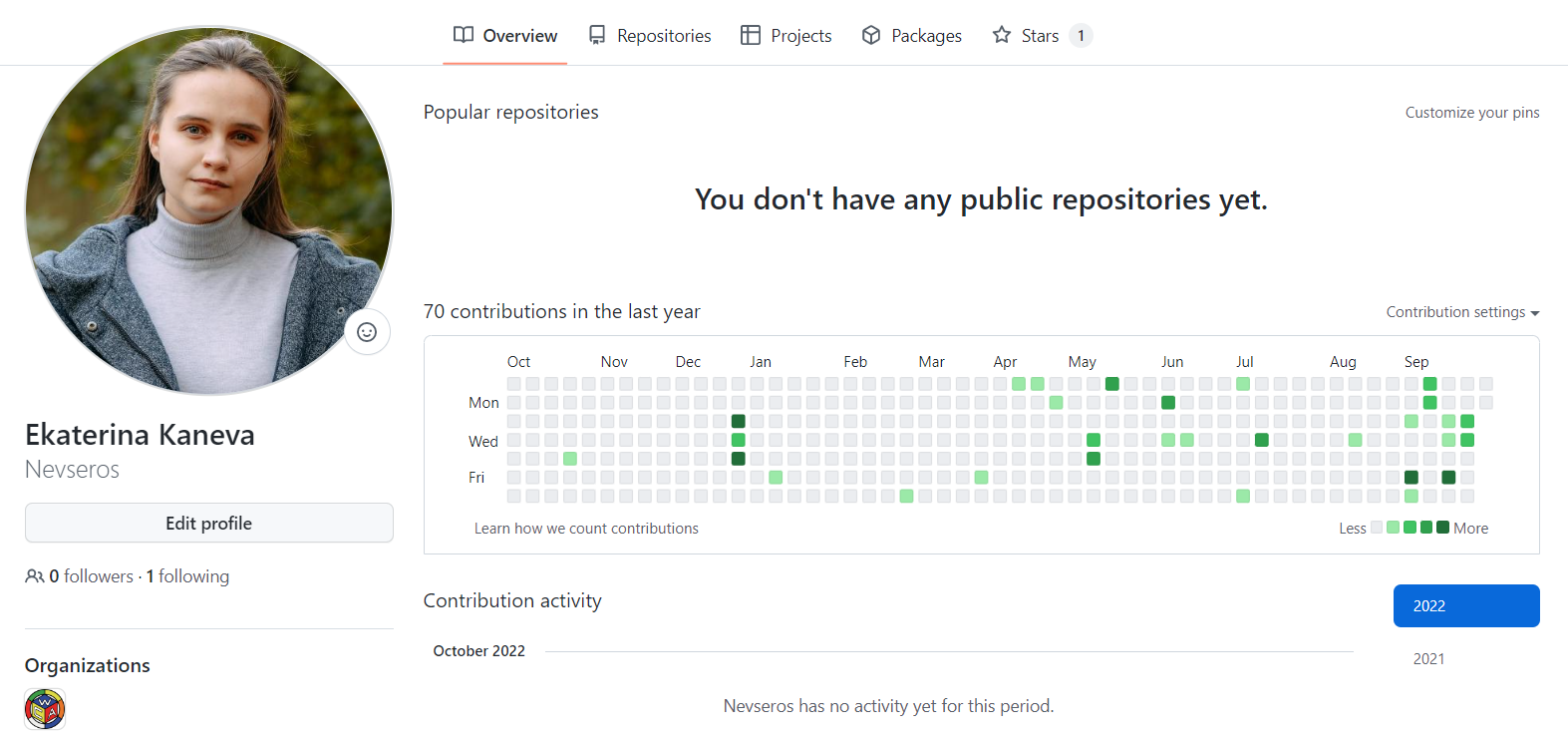


Рис. 1: Учётная запись на сайте GitHub

## 2.2 Базовая настройка git.

Сначала сделаем предварительную конфигурацию git. Откроем терминал и введём следующие команды, указав свои имя и email (рис. 2):

git config --global user.name "Ekaterina Kaneva"   
git config --global user.email "nkanevan@gmail.com"

Рис. 2: Ввод имени и адреса электронной почты.

Рис. 2: Ввод имени и адреса электронной почты.

Настроим utf-8 в выводе сообщений git (рис. 3):

git config --global core.quotepath false

Рис. 3: Настройка utf-8 в выводе сообщений git.

Рис. 3: Настройка utf-8 в выводе сообщений git.

Зададим имя начальной ветки – master (рис. 4):

git config --global init.defaultBranch master

Рис. 4: Задаём имя начальной ветки.

Рис. 4: Задаём имя начальной ветки.

Параметр autocrlf (рис. 5) и safecrlf (рис. 6):

git config --global core.autocrlf input  
git config --global core.safecrlf warn

Рис. 5: Параметр autocrlf.

Рис. 5: Параметр autocrlf.

Рис. 6: Параметр safecrlf.

Рис. 6: Параметр safecrlf.

## 2.3 Создание SSH ключа.

Для последующей идентификации на сервере репозиториев сгенерируем пару ключей – приватный и открытый (рис. 7):

ssh-keygen -C "Ekaterina Kaneva nkanevan@gmail.com"

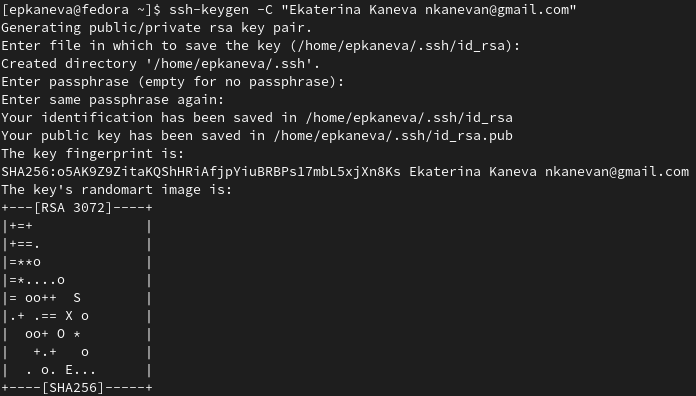


Рис. 7: Генерация SSH-ключа.

Далее, чтобы добавить новый сгенерированный ключ, авторизуемся на сайте github.org и введём новый ключ в настройках. Чтобы скопировать ключ, в консоль введём следующую команду, а затем скопируем ключ (рис. 8):

cat ~/.ssh/id\_rsa.pub

Рис. 8: Копирование ключа.

Рис. 8: Копирование ключа.

Теперь добавим ключ (рис. 9):

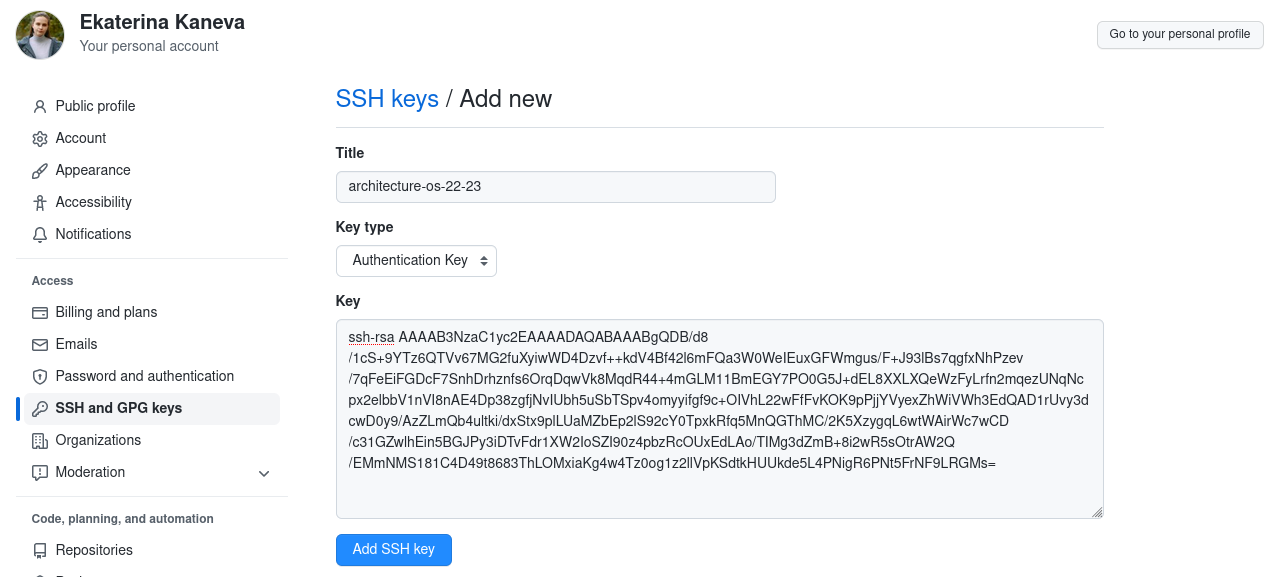


Рис. 9: Добавление нового SSH-ключа.

## 2.4 Создание рабочего пространства и репозитория курса на основе шаблона.

При выполнении первой лабораторной работы было создано рабочее пространство для предмета “Операционные системы” с помощью команды:

mkdir -p ~/work/study/2022-2023/«Операционные системы»

Используем указанный в тексте лабораторной работы шаблон для собственного репозитория (рис. 10):

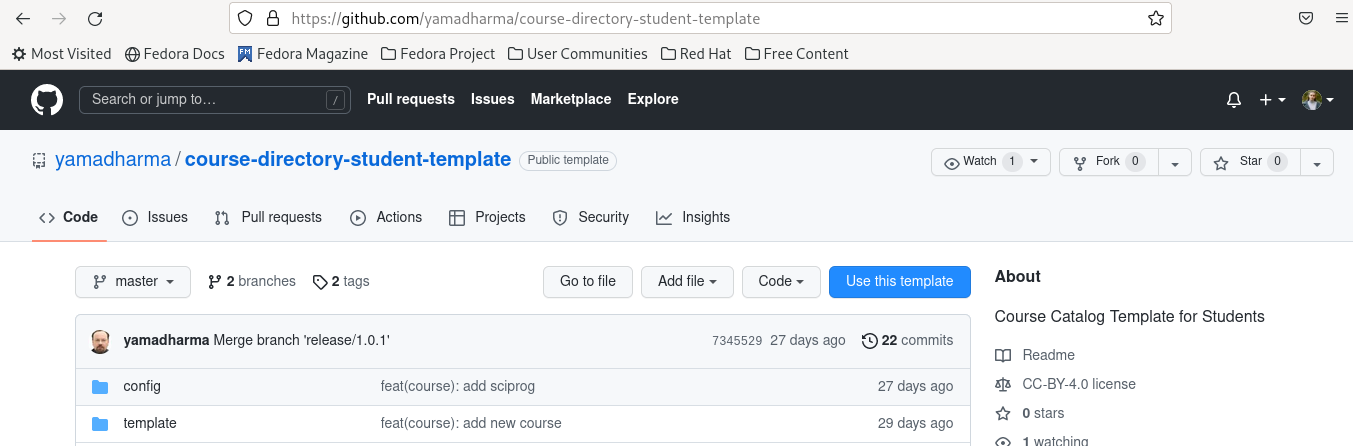


Рис. 10: Выбор шаблона для репозитория.

В открывшемся окне зададим имя репозитория study\_2022-2023\_os-intro и создадим репозиторий.

Откроем терминал, перейдём в каталог курса и клонируем репозиторий с помощью следующих команд:

cd ~/work/study/2022-2023/"Операционные системы"  
git clone --recursive git@github.com:Nevseros/study\_2022–2023\_os-intro.git os-intro

## 2.5 Настройка каталога курса

Удалим лишние файлы и создадим необходимые каталоги. Отправим файлы на сервер (всё это делалось перед выполнением первой лабораторной, поэтому скриншотов ввода команд в терминале нет:

rm package.json  
echo os-intro > COURSE  
make  
git add .   
git commit -am 'feat(main): make course structure'   
git push

Теперь проверим, что файлы действительно отправились на сервер (рис. 11):

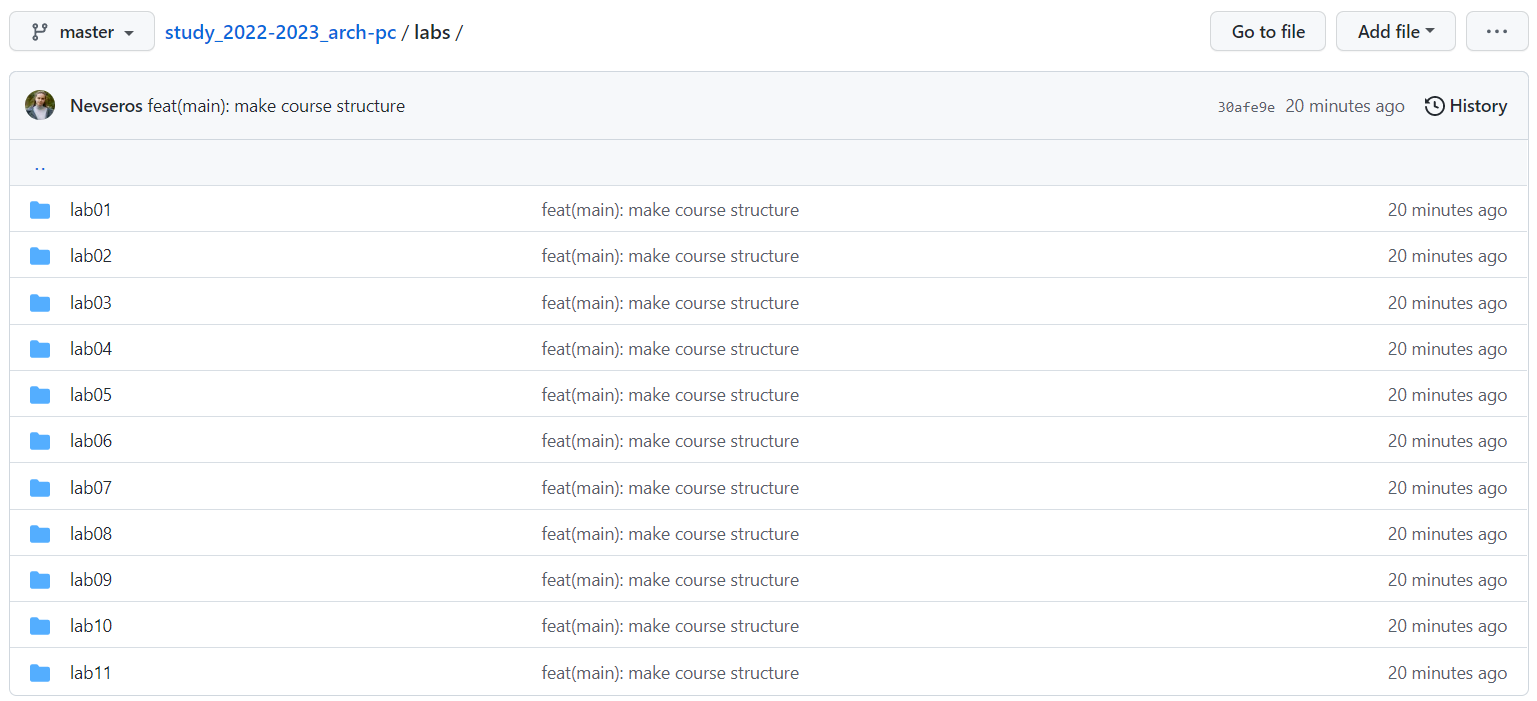


Рис. 11: Коммиты на github.

Далее началось выполнение новой части лабораторной работы - был сгенерирован gpg ключ. Для этого была введена следующая команда (рис. 12 и 13):

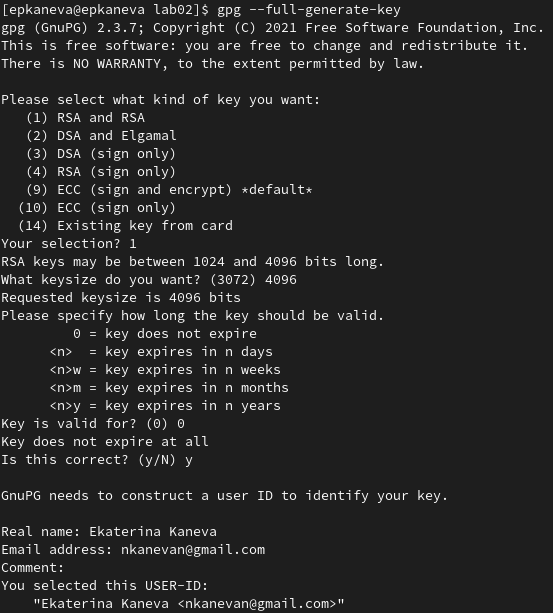


Рис. 12: Генерация gpg ключа (часть 1).

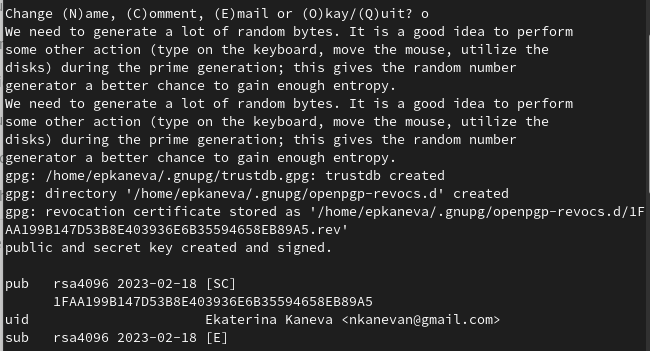


Рис. 13: Генерация gpg ключа (часть 2).

Далее для экспорта ключа была введена команда:

gpg --armor --export 6B35594658EB89A5

Где 6B35594658EB89A5 – отпечаток ключа.

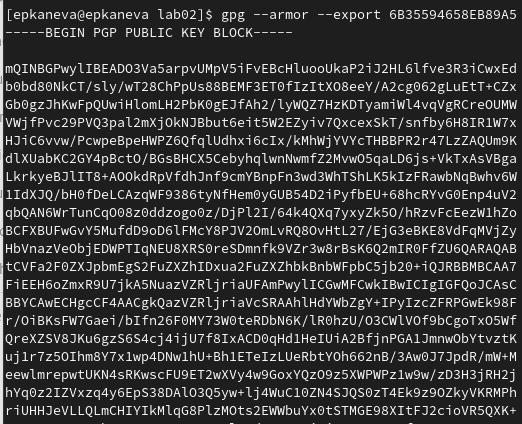


Рис. 14: Экспорт gpg ключа.

Позже этот ключ был введён на сайте github (рис. 15):

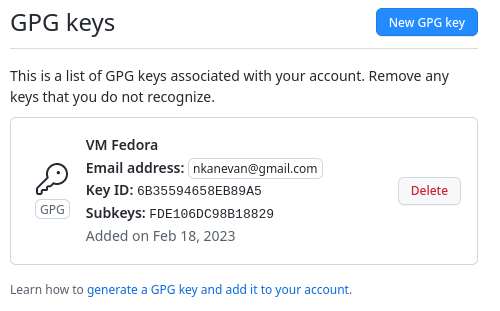


Рис. 15: Ввод gpg ключа на github.

Были настроены автоматические подписи коммитов (рис. 16):

Рис. 16: Настройка автоматических подписей коммитов.

Рис. 16: Настройка автоматических подписей коммитов.

Неверифицированные коммиты теперь выделяются (рис. 17):

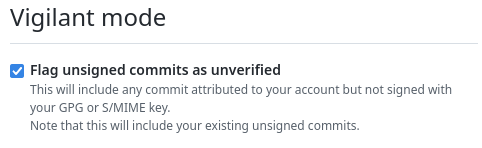


Рис. 17: Флаг на неверифицированные коммиты.

Также был выполнен вход в gh (рис. 18):

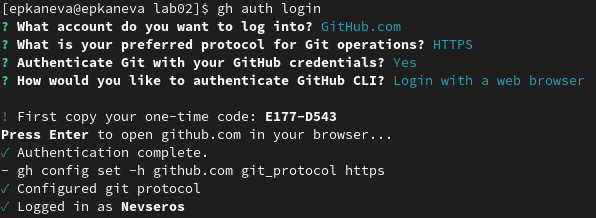


Рис. 18: Вход в gh

## 2.6 Контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

Система контроля версий – это программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов.

1. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

Хранилище – репозиторий, в нём хранятся все документы, история их изменений и другая служебная информация.

commit – добавленные изменения.

История хранит все изменения в проекте и позволяет обратиться к нужным версиям документов.

Рабочая копия – текущее состояние файлов проекта.

1. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Централизованные системы контроля версий предполагают наличие централизованного хранилища, а пользователи сами копируют все необходимые файлы к себе, добавляют изменения и отправляют их на централизованный репозиторий. Например, CVS, TFS.

Децентрализованные системы предполагают наличие у каждого пользователя своеге репозитория (а может и не одного), изменения можно получить из любого из таких репозиториев. Например, Git, Bazaar.

1. Опишите действия с VCS при единоличной работе с хранилищем.

Создаём репозиторий, по мере изменения проекта отправляем изменения (коммиты) на сервер (удалённый резозиторий).

1. Опишите порядок работы с общим хранилищем VCS.

Перед тем, как начинать изменять проект, надо получить актуальную версию. Затем можно вносить изменения, отправлять их на удалённый репозиторий.

1. Каковы основные задачи, решаемые инструментальным средством git?

Хранение файлов проекта и истории их изменения, а также удобство командной работы над проектами.

1. Назовите и дайте краткую характеристику командам git.

* Создание основного дерева репозитория:

git init

* Получение обновлений (изменений) текущего дерева из центрального репозитория:

git pull

* Отправка всех произведённых изменений локального дерева в центральный репозиторий:

git push

* Просмотр списка изменённых файлов в текущей директории:

git status

* Просмотр текущих изменений:

git diff

* Сохранение текущих изменений:
* добавить все изменённые и/или созданные файлы и/или каталоги:

git add .

* добавить конкретные изменённые и/или созданные файлы и/или каталоги:

git add имена\_файлов

* удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):

git rm имена\_файлов

* Сохранение добавленных изменений:
* сохранить все добавленные изменения и все изменённые файлы:

git commit -am 'Описание коммита'

* сохранить добавленные изменения с внесением комментария через встроенный редактор:

git commit

* создание новой ветки, базирующейся на текущей:

git checkout -b имя\_ветки

git switch -c

* переключение на некоторую ветку:

git checkout имя\_ветки

git switch

* отправка изменений конкретной ветки в центральный репозиторий:

git push origin имя\_ветки

* слияние ветки с текущим деревом:

git merge имя\_ветки

* Удаление ветки:
* удаление локальной уже слитой с основным деревом ветки:

git branch -d имя\_ветки

* принудительное удаление локальной ветки:

git branch -D имя\_ветки

* удаление ветки с центрального репозитория:

git push origin :имя\_ветки

* Переключение на старые коммиты:
* новым коммитом (можно отменить):

git revert имя\_коммита

* без нового коммита

git reset --hard имя\_коммита

1. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Например, в каком-либо проекте внесли изменения в файлы, теперь нужно их отправить на сервер. Предварительно, перед внесением изменений, притянем изменения с удалённого репозитория, потом создадим новую ветку под нашу (допустим) feature, чтобы потом открыть pull request на github (или merge request, если gitlab):

git pull  
git switch -c feature/redirect  
git add .  
git commit -m 'Add redirect route'  
git push --set-upstream origin feature/redirect

После этого в удалённом репозитории появится вохможность открыть pull/merge request. Там, по необходимости, нужно будет дождаться фидбэка от других контрибьюторов, решить конфликты веток, если таковые есть, а потом влить изменения.

1. Что такое и зачем могут быть нужны ветви (branches)?

Ветви – параллельные участки истории. Нужны, чтобы работать над разными заданиями одновременно, а также чтобы удобнее было работать нескольким людям над одним проектом, чтобы можно было открывать pull/merge request’ы и получать ревью кода.

1. Как и зачем можно игнорировать некоторые файлы при commit?

Во время работы над проектом могут появляться ненужные файлы (например, какие-то временные файлы создаваемые редакторами или объектные файлы, создаваемые компиляторами). Их можно игнорировать с помощью .gitignore (специальный файл в репозитории). Можно также использовать команду git reset HEAD имя\_файла, чтобы исключить его из текущего коммита (после git add).

# 3 Выводы

Изучили идеологию и применение системы контроля версий. Приобрели практические навыки работы с системой git.